# Understanding Computing in a Hybrid World

## On the Undergraduate Curriculum Front-End Development

Laura Benvenuti
Hogeschool van Amsterdam
Amsterdam, the Netherlands
l.benvenuti@hva.nl

Erik Barendsen
Radboud Universiteit,
Open Universiteit
Nijmegen, the Netherlands
e.barendsen@cs.ru.nl

Gerrit C. van der Veer
Dalian Maritime University
Dalian, China
gerrit@acm.org

Johan Versendaal
Open Universiteit
Heerlen, the Netherlands
johan.versendaal@ou.nl

## ABSTRACT

Computing is an interdisciplinary field that can be approached from different points of view. Each point of view has its goals, aims and fundamental assumptions. This makes computing a complex discipline. Moreover, new computing disciplines appear regularly. With the trend that ICT-professionals should have non-ICT competences as well, and non-ICT-professionals should have ICT-competences, new computing curricula are often *hybrid* in nature. As a hybrid computing curriculum cannot cover the full range of computing, it is interesting to investigate the 'computing part' of such curricula.

Our analysis framework consists of three elements: the curricular components 'goals and objectives' and 'instructional strategies', and the underlying epistemological view on the discipline ('cultural styles'). Taking a historical perspective, we describe the origins of the ACM/IEEE Curriculum Recommendation series. We discuss the three main cultural styles of computing: theoretical, scientific and engineering.

Observing that in a curriculum the above elements should be aligned, we present three trade-offs for the case of hybrid computing curricula. We apply our results to two concrete examples, *Liberal Arts and Computer Science* and *Front End Development.* Based on our investigation, we formulate recommendations for designers of hybrid computing curricula. We recommend, for example, discussing disciplinary boundaries and resulting trade-offs explicitly while designing and documenting curricula.


*SIGCSE'18, February 21-24, 2018, Baltimore, MD, USA.*

ACM ISBN 978-1-4503-5103-4/18/02...$15.00.
https://doi.org/10.1145/3159450.3159532

## CCS Concepts

• Social and professional topics ~ Computing education programs
• Social and professional topics ~ History of computing

## Keywords

Interdisciplinary programs (CS+X); Hybrid undergraduate curricula; Cultural styles in computing; Front End Development

**ACM Reference format:**

Laura Benvenuti, Erik Barendsen, Gerrit van der Veer and Johan Versendaal. 2018. Understanding computing in a hybrid world. In *Proceedings of SIGCSE'18, February 21–24, 2018, Baltimore, MD, USA.* 6 pages. https://doi.org/10.1145/3159450.3159532

## 1 INTRODUCTION

New computing curricula often are hybrid in nature. Since the introduction of the World Wide Web, new areas, commonly referred to as Creative Technologies, have gained importance. Educational programs appear, meant to foster these developments by stimulating crossovers between computing and non-technological sectors as fashion, health etc. The corresponding curricula only partially concern computing, since a considerable part is devoted to sector specific topics. Designing curricula for hybrid contexts poses interesting questions. Which part of computing is relevant for hybrid programs, in how far should we consider graduates of these curricula as computing professionals? The answer to these questions is controversial.

Recent research on the Dutch labor market [7] observes that boundaries between Information and Communication Technology (ICT) and other sectors are fading away. It refers to the European e-Competence Framework (eCF) [8] that divides specific ICT competences in primary ICT skills, as software development, and secondary ICT skills supporting primary skills, like information security strategy development of user support. Specific ICT competences are increasingly requested in other professions than the computing professions listed in eCF. If professional profiles were classified by the required ICT competences, argues the Dutch report, the computing professions would double. The report

predicts a shortage of professionals having primary ICT skills in the immediate future, in particular a shortage of software developers. It also warns against the growth of hybrid undergraduate computing curricula, where 'hybrid' is defined as: "programs that primarily concern other areas than computing". According to the Dutch researchers, hybrid curricula focus too much on secondary ICT skills to comply with the industry's demand.

We question the researchers' conclusions about the education of software developers. But we do acknowledge that designing computing content for hybrid setting is a challenge. This paper strives at understanding computing, to scaffold choices, necessary to design hybrid curricula. Which part of computing is relevant in hybrid contexts, and why?

In our pedagogical analysis we consider two central curricular themes (cf. [1]), namely *Goals and objectives of computing curricula* and *Instructional strategies specific for teaching computing*. The underlying cultural or epistemological view on the discipline [19] is a third theme, which we refer to as *Cultural styles*. In a balanced curriculum, the above three themes are *aligned*, that is, goals and instructional strategies fit together and are consistent with an underlying cultural style.

We will conduct our investigation from a historical perspective. In Section 2 we will sketch the preamble for the joint ACM/IEEE series of curriculum guidelines, to better understand the international guidelines for the undergraduate computing curricula, in particular with respect to goals and objectives of computing curricula.

In Section 3, we will further examine the interdisciplinarity of computing. Besides a comparison of the different cultural styles in computing, we will discuss the implications for computing education, especially concerning instructional strategies.

We explore the challenges for hybrid curricula in terms of trade-offs between the three themes in Section 4. In Section 5 we apply our analysis to two particular hybrid curricula, *Liberal Arts and Computer Science* and *Front End Development*. We conclude with a summary and recommendations in Section 6.

## 2 COMPUTING CURRICULUM GUIDELINES

North America has a tradition of dialogue between professionals covering different roles in the computing community: scholars, professionals, engineers, and theoreticians. This is remarkable; in the Netherlands, the computing community was shaped by debates and is still divided today [4]. In this section, we will explore the preamble to the Joint Task Forces on Curriculum Recommendation, that were initiated in North America but operate worldwide today.

The usage of programmable computing machines for practical purposes took off with the Second World War in the United States, Germany and Great Britain. But the European computers did not survive the war. The German Z3 was destroyed in an allied bombardment of Berlin in 1943; the British Colossi were considered highly classified and were dismantled for that reason. Things went differently in the U.S.A. The ENIAC was announced to the press in 1946. In the summer of that year, the Pentagon organized lectures about the construction of digital electronic computers, and invited computing professionals from the United States and Great Britain [12]. The dissemination of the ideas underlying computing construction starts here.

Societies for computing professionals were quickly established: one of the precursors of IEEE (the Subcommittee on Large-Scale Computing of the American Institute of Electrical Engineers) in 1946, followed by ACM in 1947. American organizations of computing professionals have collaborated since the early days [9] by sponsoring joint conferences.

Tedre [20] draws a detailed picture of the debates that have shaped the discipline in the second half of the 20th Century. Until the 1960s, mathematicians collaborated with engineers to realize computing machines and languages to program them. These activities resulted in artifacts that have an impact on the world, an impact that sometimes is measurable, but not always. Should computing be considered an art or a science? Should practical topics as programming be taught in academy; is computing an academic discipline at all?

### 2.1 A theoretical approach

The first edition of the ACM curriculum recommendations, published in 1968, took a stance in this debate. Computer Science (CS) had to be considered an academic discipline studying information structures and processes [20]. Its core issue was handling abstraction. Mathematics provided for theoretical foundation of the discipline. In these days, computer scientists were accommodated in Departments of Mathematics or in Departments of Electronic Engineering. Training for the positions of programmers could be supplied by technology programs, vocational education or junior colleges.

### 2.2 The software crisis

The theoretical approach appeared not to be able to solve all the problems of the booming discipline. In the late 1960s, software complexity increasingly had become problematic. Complexity did not only arise from the parts of a system – object of study in the Computer Science (CS) departments - but from the connections between these parts [20]. This software crisis uncovered a lack of methodology for understanding and controlling software systems that were too big for single programmers. Software Engineering (SE) emerged, a discipline that considers software development as the effort of a team whose members might have contrasting intentions.

The update of the ACM Curriculum in 1978 brought programming back in the academic curriculum. With the software crisis, the focus of computing had shifted from a discipline studying information structures into an application-centered discipline [20]. Computing had gained independence from mathematics and engineering, but a clear definition of the discipline had been lost.

Curriculum '78 has been criticized for too strongly identifying computing with programming. In their objection to that view, Ralston and Shaw [14] stated that mathematics lays a foundation under both science and engineering, and that understanding of any of these disciplines is impossible without understanding

mathematics. Also, the mathematic foundations of computing are stable, where specific skills rapidly become obsolete. Giving students a background in mathematics would protect them from this obsolescence, pleaded Ralston and Shaw.

### 2.3 The Snowbird conferences in the 1980s

Starting with 1972, the Heads of CS Departments offering PhD in the USA and Canada held biennial meetings in Snowbird, Utah. Members of government and representatives from the computer industry attended these meetings, which evolved towards meetings of the North American computing community.
The 1980 Snowbird Report [5], "A discipline in crisis", depicts a worrisome situation of Computer Science Departments. The computing field was growing explosively. Universities were overburdened by the growth of undergraduate enrollments. There was no time for supervision of graduate students. The best students were hired by industry, which caused a shortage of PhDs. The situation threatened the ability to conduct basic research in Computer Science.

As an answer to the problems described in "A discipline in crisis", dialogues were improved between academia, the industry, government agencies and professional organizations [22]. The 1984 report (Future Issues in Computer Science) [17] addresses the question of the lack of agreement about the accreditation of undergraduate programs. ACM and IEEE had started investigating the desirability of cooperating in the accreditation process at the 1982 Snowbird conference [22]. By 1984, the professional organizations had created a Computer Science Accreditation Board (CSAB); the volunteers necessary to administrate the CSAB were provided by academy.

One of the aims of CSAB was: providing a more common view of a core description, at least for the undergraduate curriculum. ACM and IEEE formed a joint task force to describe the intellectual substance of the computing field in 1985[6]. A joint task force for the definition of the undergraduate curricula was formed in 1988 [15].

### 2.4 The ACM/IEEE curriculum reports

The Task Force, chaired by P. Denning, presented its final report "Computing as a Discipline" in 1989. The Task Force uses the phrase *discipline of computing* as an umbrella term for scientific and engineering aspects of computing. It starts with stating having extended its task to both Computer Science and Computer Engineering because it had concluded that *no fundamental differences exist between the two fields in the core material.* [6] The report distinguishes three major paradigms, or cultural styles, for the computing discipline: theory (rooted in mathematics), abstraction (rooted in the experimental scientific method), and design (rooted in engineering). The three cultural styles are fundamental for the discipline, which is described as a blend of interaction among theory, abstraction and design.

"Computing as a discipline" addresses the question, in which area of computing majors should be competent. The answer is cautious: discipline-oriented thinking should be the primary goal of every curriculum for computing majors, based on solid mathematical foundations. Teaching paradigms emphasizing inquiry and orientation in the computing literature are recommended, instead of lectures presenting answers.

"Computing as a discipline" pleads against a strong identification of computing with programming. It also states that every computing major should be competent in programming. The Task Force remarks, that the distinctions between the computing disciplines are embodied in the differences between programming languages, and recommends programming languages to be treated as a vehicle to access these distinctions.

Since 1989, the ACM/IEEE has jointly formed Task Forces that have published and iterated overview reports with recommendations for computing curricula. They appeared in 1991 and 2005. These reports have a triple ambition. They aim at (1) training the professionals requested by industry, (2) training the students' intellectual skills necessary to find employment, enter the Master level and keep up with future developments, and (3) granting the development of the discipline of computing.

Over the years, the amount of sub disciplines has narrowed to 5 [2]: Computer Science, Computer Engineering, Information Systems, Information Technology, and Software Engineering. Undergraduate curriculum recommendations for each of these areas were published in 2005. Specific committees of ACM and IEEE representatives worldwide have updated them since then. CS2013 applies to Computer Science, one of the sub-disciplines of computing. It falls outside the scope of this investigation.

### 2.5 Hybrid computing curricula

Until the 1970s, computers were used in professional setting in the industry, in academia and in big governmental institutions. The first computer users were programmers, but the user base changed as the costs of hardware lowered [20]. New interdisciplinary fields of study emerged, as Management Information Systems, combining computing topics with topics from other disciplines.

Today, all computing curricula are "hybrid" to a certain extent. Non-computing topics as Risk Management or Interpersonal Communication are taught in most undergraduate computing programs. But there also is a "core of computing", common to all these programs. The last ACM/IEEE Overview Report, CC2005 [2], provides 10 knowledge areas and 40 topics every Computing Curriculum should cover. The weight of these topics in single curricula can differ, and some topics might not be covered at all, but the clear intention of the Report is to describe a core, common to the five sub disciplines of computing.

Today, there also are undergraduate curricula covering a significant part of the core of computing, but omitting another significant part. Some examples are: Medical Information Systems [7], Business Analytics [7], Liberal Arts and Computer Science [10]. We will adopt the Dutch researchers' definition and will indicate as "hybrid" these curricula dedicating less than half of their program to topics belonging to the core of computing.

## 3 CULTURAL STYLES IN COMPUTING

The Task Force's view on three cultural styles of computing [6] was not new. In 1969, Peter Wegner had held a lecture at the

annual meeting of the American Association for the Advancement of Science. Its title was: Three computer traditions [21]. Wegner objected to a single view of computing. CS, argued Wegener, is *in part a scientific discipline concerned with the empirical study of a class of phenomena, in part a mathematical discipline concerned with the formal properties of certain classes of abstract structures, and in part a technological discipline concerned with the cost-effective design and construction of commercially and socially valuable products.*

In a recent article [19], Tedre and Apiola discuss the different cultural styles of computing, or (in their terminology) three traditions of computing. According to the authors, these three traditions fulfill different roles in the development of the discipline of computing. The scientific tradition copes with the fundamental problem: does an abstract model fit the world? The engineering tradition copes with the question, how to translate abstract models into working artifacts. The theoretical tradition aims at building a coherent abstract framework supporting understanding of notions as algorithms, complexity, data structures. All authors agree upon the importance of intertwining the traditions in the computing practice.

Despite the statement in the 1989 report, that "no fundamental difference exists between the [...] fields in the core material", founding the discipline on three traditions is a challenge. Tedre and Apiola [19] analyze the differences between the three cultural styles, and the implications of these differences implications for basic (K-12) computing education. They state that, however intertwined and overlapping, the three cultural styles are fundamentally different in their aims, in the status of the knowledge they pursue and in their methodological views.

The aim of the theoretical tradition is to produce coherent structures, in order to describe algorithms and cope with complexity. The scientific tradition aims at understanding the world by modeling phenomena (such as the weather, the mind) and testing the accuracy of the models. The engineering tradition aims at changing the world, by producing artifacts that fulfill social needs or desires.

In the theoretical tradition, knowledge is considered independent of what people may think. It is considered universal and is validated in its theoretical context. Also the scientific tradition claims to aim at value-free knowledge, which should be descriptive. This is opposed to the engineering tradition, in which knowledge is partially value-free (where it concerns physical constraints) and partially value-laden (where it concerns social needs and desires). The Body of Knowledge in the engineering tradition is partially descriptive and partially normative (know-how).

The theoretical tradition analyzes ideas. Propositions should be proven. The scientific tradition observes the world. Claims should be sustained by empirical results. The engineering tradition acts. It evaluates tangible products. Engineers must be able to act, even without having sufficient information to fully sustain the design of the artifact.

Tedre and Apiola's paper concerns K-12 computing education. The authors plead for aligning learning objectives with the corresponding computing traditions (expressed in pedagogic approaches and educational resources), because a mix would not result in effective educational interventions. If the learning objectives of the school concern construction processes, assessment tasks should not be propositional (quizzes), but procedural (assessment of design artifacts), because they better fit in the related tradition, which in this case is engineering. The authors stress that educators should understand all the traditions of computing. They warn for bias induced by hidden ethos elevating one of the traditions above the others, both inside schools as in university departments educating teachers.

Tedre and Apiola reflect on the role of computing traditions in educational design. The authors recommend matching the tradition with the learning objectives of educational units. In the end, it is an argument about the efficacy of educational interventions. It does not concern the learning objectives themselves, nor the goals and objectives of computing curricula.

## 4 DISCUSSION: CURRICULAR TRADE-OFFS

In this section we will reflect on hybrid computing education using the three pedagogical themes described earlier. Alignment of the three themes in combination with the limited space for computing in a hybrid curriculum gives rise to challenges. Below we will discuss three of these challenges: representing all cultures (cultural styles), fostering of discipline oriented thinking (goals and objectives) and the role of mathematics (goals and objectives, again). In our opinion, the trade-offs involved with curricular choices ought to be made explicit by authors of hybrid curricula. In Section 5 we will discuss two cases in more detail.

### 4.1 Incorporation of the three cultural styles

In 1989, the Task Force on the Core of Computer Science described a common core of the discipline of computing. Content from each cultural style should be treated by all undergraduate computing curriculums. To prepare students for the future –which was uncertain— the 1989 recommendations included *inquiry-based* learning activities and orientation in the computing literature, instead of lectures presenting answers.

According to Tedre and Apiola, the three cultural styles not only entail different approaches to the discipline, but also respect different epistemological values. Should value-laden knowledge be accepted (as it is in the engineering culture) or should knowledge always be value-free? Should ideas be analyzed (theoretical culture) or evaluated empirically (scientific culture)?

An inquiry-based curriculum appears not to be neutral from this perspective. There is a risk that Institutions (schools, academies, departments) prefer one culture, one approach to research above others. Even an inquiry-based educational approach can canalize students' understanding of the discipline, towards values, consistent with the method of inquiry.

A possible direction for a solution is given by "Computing as a discipline" in stating that "*most of the distinctions in computing are embodied in programming notations*". It suggests using differences between programming languages as a vehicle to discuss differences between approaches to computing. We will return to this in Section 5.2.

## 4.2 Fostering discipline oriented thinking

The ACM/IEEE curricula are written from a triple ambition: to serve the students, serve the industry and grant the development of a (unified) discipline. In the light of Tedre's and Apiola's findings, we are not optimistic about the possibility of fulfilling all the ACM/IEEE curricular ambitions by any of the undergraduate computing curricula. Especially the aims concerning overview of the discipline and employability appear to be scarcely compatible with each other. Mixing content from cultural styles will not automatically grant overview of the discipline, since knowledge is also rooted in the understanding of research methods. Nor will an investigative approach to education necessarily support discipline oriented thinking.

Employability and capability to keep up with future developments often suppose up-to-date knowledge in one of the sub disciplines and related research skills. Once acquainted with one of the cultural styles, the graduate is likely to pursue a career path compatible with it, reinforcing the chosen orientation.

If granting overview of the discipline is challenging for hybrid curricula, explicating boundaries becomes necessary.

## 4.3 The role of mathematics

Both the 1989 Task Force and the authors of "Computing as a discipline" recognized the importance of a solid mathematical foundation to foster discipline oriented thinking.

We partially agree on the overall importance of mathematical thinking for computing professionals. Yes, the theoretical cultural style does support discipline oriented thinking, because it provides a common language to discuss computing constructs. It certainly does support formal reasoning. But Mathematics too is a vast discipline. Different branches of computing rely different branches of Mathematics. A course on continuous mathematics supports skills that are relevant to computer engineers, not necessarily to information scientists. The existence of one form of mathematical training, apt to foster overall understanding of computing, is debatable.

# 5 HYBRID CURRICULA: TWO CASES

In this section we will apply our findings to two existing hybrid programs: Liberal Arts and Computer Science and Front End Development, respectively. For each of these we will examine our pedagogical themes and the appearance of trade-offs.

## 5.1 Liberal Arts and Computer Science

The Liberal Arts and Computer Science (LACS) curriculum (USA) [10], has its origin in a reaction to ACM's Curriculum '78, a strongly engineering oriented curriculum [3][20]. Liberal Arts undergraduate programs traditionally emphasize intellectual growth of students, rather than preparing them for a specific career. In 1984, a consortium of small Liberal Arts colleges offering computing degrees, was formed to discuss the problems these colleges had with the implementation of Curriculum '78. The meetings resulted in the first Model Curriculum for a Liberal Arts Degree in Computer Science (LACS), which appeared in 1986.

LACS offers a Bachelor of Arts degree. Roughly 40% of the curriculum is dedicated to Computer Science, 5-10% to other science courses and the remaining 50-55% to humanities and social sciences [10]. LACS is a hybrid computing curriculum.

It is interesting to see, what the LACS curriculum guidelines states about the role of mathematics. Mathematical education is considered part of the Computer Science curriculum, for several reasons. Computing relies on mathematical objects (as sets, relations) and mathematical reasoning (logic, algorithms, correctness proofs); mathematical tools as probability and statistics allow analysis of software [10].

*5.1.1 Cultural styles* LACS chooses a generalist approach and de-emphasizes specific technical details. It acknowledges the importance of mathematics in this – hybrid – curriculum. In terms of Tedre's and Apiola's work, the theoretical and the scientific view are combined in the LACS curriculum, but the engineering viewpoint is somehow overshadowed.

*5.1.2 Goals and objectives* LACS strives to train generalists. The curriculum de-emphasizes specific technical details; its ambition is to develop student's intellectual skills. The objective of the curriculum is to support durable intellectual growth, combined with (generalist, discipline oriented) helicopter view.

*5.1.3 Trade offs* The LACS curriculum explicitly makes concessions to its ambitions towards software development.

## 5.2 Front End Development

We will refer to the discipline, specialized in the development of multimodal user interfaces as Front End Development (FED) and the corresponding professional figures, Front End Developers (FEDs):.

"A front-end developer specializes in building the front end, or client-side, of a web application, which encompasses everything that a client, or user, sees and interacts with. Front-end development is all about what's *visible* to the user." [16] .

There is not yet consensus about an undergraduate curriculum for FED. In Amsterdam, FED is a major of the undergraduate curriculum Communication and Multimedia Design, in the domain Creative Technologies. FEDs are professional users of client-side technology. They work in interdisciplinary teams in developing projects. FEDs produce software that interacts with APIs, services and other ready-to-use software components, written by other computing professionals.

*5.2.1 Cultural styles* FED is teamwork and aims to produce maintainable software products. For these reasons, we tend to include FED among the disciplines emphasizing primary ICT skills as described in section 1. Still, not all of computing is relevant for FEDs. Most of the Front End Developer's work is visible to the user and can be tested directly. There is no need for correctness proofs in FED. FEDs need Mathematics to interpret quantitative research data. Acquaintance with basic abstract structures as sets and enumerations is needed, since every Front End communicates with other computer systems through abstract interfaces. In how far modeling skills are necessary for FEDs is subject to discussion. In terms of Tedre's and Apiola's work, the engineering and the scientific aspects of computing are emphasized by this program, at the expenses of theoretical aspects.

*5.2.2 Goals and objectives* FED is characterized by focus on craftsmanship, in the context of interdisciplinary developing teams and rapid technologic evolution. It strives at training competent manpower for the discipline of FED, who will be able to to comply with future developments in FED.

*5.2.3 Instructional strategies* Front-end developing languages are introduced stressing their aims and their boundaries, conform the Task Force's remark on programming languages: "Software, written with HTML, performs behavior that is visible. It is not necessary to express intended outcomes in formal statements; the software can be tested. To support interaction with abstract agents, professional HTML code should respect conventional semantics. These are defined by a political process". This supports future FEDs' understanding of the scope of their craftsmanship, helps them to follow the evolution of their discipline, and to participate in that process.

*5.2.4 Trade offs.* The choice for an engineering-based FED curriculum, at the expense of the theoretical content, has consequences. Graduates are not all-round software developers. Their sphere of activity is limited to the user interface. FEDs are trained in making Front-End artifacts, not in developing Front-End technology. They use and research the possibilities of existing technology. The question, whether FEDs should develop simple APIs is still open.

FED graduates are granted access to the Master's level in hybrid computing disciplines as Digital Communication and Media, not to Master's programs in Software Engineering.

## 6 CONCLUSIONS

We investigated the possibility to type hybrid computing curricula. Our conclusion is: refer to the complex epistemological background of computing to type hybrid curricula. It is inevitable for designers of hybrid curricula to make choices. This approach gives insight in the related trade-offs; it is relevant for both educational institutions and accreditation boards.

We do think that some of the hybrid computing curricula can train developers, although bound to specific contexts, like FEDs. Though, it is fundamental to describe and stress the boundaries of such programs, and of the corresponding professions. Defining and tuning undergraduate computing curricula is not only a matter of training manpower requested by industry. We saw that the field of computing was shaped by a joint effort of industry, academy and governmental institutions, and urge designers of hybrid curricula, to take responsibility for the definition of new disciplines. Trade-offs, involved by curricular choices, should explicitly be discussed in broad communities including: related computing disciplines in academy, the industry and the public authorities.

Following Tedre's and Apiola's observations, we also recommend that all educators of computing topics understand the complex nature of computing. This applies in particular to those lecturing in hybrid curricula, a condition that is not always met. We address educators, to warn them against a sloppy approach to computing: (1) lecturing the outcomes without the method would not result in durable knowledge; (2) lecturing technology without mentioning its trade-offs would not allow students to position themselves as professionals in the dynamic field of computing; (3) lecturing the method without the philosophical motivations would mean not taking students seriously enough to invite them to join the conversation about their own professional future.

## REFERENCES


[1] Van den Akker, J. (2003). Curriculum perspectives: An introduction. In: Van den Akker, J., Kuiper, W., Hameyer, U. (eds.) *Curriculum landscapes and trends* (pp. 1-10). Springer Netherlands.

[2] ACM/IEEE. (2005). Computing Curricula 2005. The Overview report, Retrieved from https://www.acm.org/education/curricula-recommendations, 10.6.2016

[3] Bruce, K. B., Cupper, R. D., & Drysdale, R. L. S. (2010). A history of the liberal arts computer science consortium and its model curricula. ACM Transactions on Computing Education (TOCE), 10(1), 3.

[4] Dael, R. L. H. V. (2001). 'Iets met computers': over beroepsvorming van de informaticus. Delft: Eburon.

[5] Denning, P. J., Feigenbaum, E. A., Gilmore, P., Hearn, A. C., Ritchie, R. W., & Traub, J. F. (1981). A discipline in crisis. Communications of the ACM,24(6), 370-374.

[6] Denning, PJ, DE Comer, D. Gries, MC Mulder, A. Tucker, AJ Turner, and PR Young (1988). "Computing as a discipline." Communications of the ACM 32, no. 1 (1989): 9-23.

[7] Dialogic & Matchcare (2016). Digitaal vakmanschap, van de ICT arbeidsmarkt naar de arbeidsmarkt voor ICT'ers, retrieved from https://www.nederlandict.nl/news/tekort-aan-developers-neemt-toe/2.11.2016

[8] eCF, European e –Competence Framework 3.0, retrieved from http://www.ecompetences.eu, 02.6.2016

[9] Joint Computer Conferences, retrieved from https://en.wikipedia.org/wiki/Joint_Computer_Conference , 18.3.2016

[10] Liberal Arts Computer Science Consortium. (2007). A 2007 model curriculum for a liberal arts degree in computer science. Journal on Educational Resources in Computing (JERIC), 7(2), 2.

[12] Moore School Lectures, retrieved from https://en.wikipedia.org/wiki/Moore_School_Lectures, 12.01.2016

[13] Ralston, A. (1981). Computer science, mathematics, and the undergraduate curricula in both. The American Mathematical Monthly, 88(7), 472-485.

[14] Ralston, A., & Shaw, M. (1980). Curriculum'78 - is computer science really that unmathematical?. Communications of the ACM, 23(2), 67-70.

[15] Reilly, E. D. (2004). Concise encyclopedia of computer science, 4th edition (p.291). John Wiley & Sons.

[16] http://www.skilledup.com/articles/web-developer-job-descriptions-skills-they-require retrieved 25-5-2016

[17] Tartar, J., Arden, B., Booth, T., Denning, P., Miller, R., & Van Dam, A. (1985). 1984 Snowbird Report: Future Issues in Computer Science. Computer, 18(5), 101-105.

[19] Tedre, M., & Apiola, M. (2013). Three computing traditions in school computing education. In: Kadijevich, D. M., Angeli, C., & Schulte, C. (eds.). Improving computer science education. Routledge, 2013, CH. 7.

[20] Tedre, M.: lecture Notes on the Philosophy of Computer Science, retrieved September, 3rd 2015 from http://cs.joensuu.fi/~mmeri/teaching/2007/philcs/

[21] Wegner, P.(1970) Three computer traditions: Computer technology, computer mathematics, and computer science. Advances in computers, 10, 7-78.

[22] Yau, S. S., Ritchie, R. W., Semon, W., Traub, J., Van Dam, A., & Winkler, S. (1983). Meeting the Crisis in Computer Science. Communications of the ACM, 26(12), 1046-10